Ivy : un bus logiciel au service du développement de prototypes de systèmes interactifs

Marcellin Buisson, Alexandre Bustico, Stéphane Chatty, Francois-Régis Colin, Yannick Jestin, Sébastien Maury, Christophe Mertz, Philippe Truillet

Ivy : Un bus logiciel au service du développement de prototypes de systèmes interactifs

Marcellin Buisson (+,°), *Alexandre Bustico (+), Stéphane Chatty* (*), *Francois-Régis Colin (+), Yannick Jestin* (+), *Sébastien Maury (+), Christophe Mertz* (+,°), *Philippe Truillet* (+,×)

(+) CENA
7, avenue Ed. Belin
31400, Toulouse, France
{buisson, bustico, fcolin, jestin, maury, mertz, truillet}@cena.fr

(°) Transiciel Technologies.
13, rue Villet
31400 Toulouse, France
{marcellin.buisson, christophe.mertz}@tso.transiciel.com

(*) IntuiLab
Prologue 1,
La Pyrénéenne
31312 Labège, France
chatty@intuilab.com

(×) IRIT
UMR CNRS 5505
118, route de Narbonne
31062, Toulouse
France
truillet@irit.fr

RESUME

Ce document présente l'expérience acquise au cours du développement et de l'utilisation du bus logiciel Ivy, dans un cadre de prototypage de systèmes interactifs pour le contrôle du trafic aérien. Après une description du principe de fonctionnement de ce système, nous verrons comment cet outil a pu influer sur notre approche de problématiques IHM spécifiques comme la multimodalité, l'interaction répartie ou la mobilité. L'accent est porté sur les services rendus par ce bus pour le développement rapide de systèmes interactifs « légers », facilement intégrables dans un banc de démonstration et basés sur la logique des langages de script. En présentant cet outil que nous utilisons depuis maintenant cinq ans, nous espérons partager ici une expérience utile pour la conception de futures architectures de systèmes interactifs à des fins de recherche prospective.

MOTS CLES : Architecture des systèmes interactifs, outils de prototypage, bus logiciel.

ABSTRACT

This paper focuses on the experience we have acquired by developping and using the Ivy software bus to design human-computer interactive prototypes for Air Traffic Control. We list its properties for designing and developing light and powerful HMIs. By presenting our experience, we hope it will foster the design and development of the future architectures of interactive systems for research needs.

IHM 2002, November 26-29, 2002, Poitiers, FRANCE.

KEYWORDS : Interactive systems engineering, prototyping tools, software bus.

INTRODUCTION

Concevoir des systèmes interactifs expérimentaux est une tâche rendue complexe par l'hétérogénéité des environnements logiciels et matériels utilisés pour leur développement. Bien souvent, des difficultés apparaissent pour connecter les pilotes de périphériques aux boîtes à outils multimodales dédiées à l'input et aux boîtes à outils dédiées au rendu graphique et à l'animation. Or les environnements de développement sont pour la plupart très cloisonnés et limités à des plateformes logicielles ou matérielles spécifiques. Comment faire communiquer plus efficacement tous ces systèmes ? Deux tendances se dégagent : choisir un seul environnement de développement cohérent mais au prix de potentialités restreintes ou alors opérer sur différentes plateformes mais avec les problèmes de communications et d'intégration de ces développements au sein d'un système interactif performant. C'est ce type de dilemme auquel a été confrontée il y a quelques années notre équipe, et c'est en partie en réponse à ces questions qu'est né Ivy.

L'un des objectifs initiaux est la recherche de solutions viables pour unifier les développements tout en maintenant des cycles de prototypage courts. Il est en effet essentiel que notre environnement de développement favorise l'obtention d'un retour d'expérience sur les fonctionnalités des prototypes, le suivi d'un cycle de développement en spirale et incrémentiel décrit dans [1] et [6].

L'autre objectif est de pouvoir développer des interfaces multimodales intégrant voix et gestes sur écrans tactiles et de les intégrer au sein d'un banc de démonstration intégrant nos prototypes. Ce dernier objectif impliquait

de faire dialoguer des PC Windows, Linux et des Macs et d'utiliser des langages différents.

En 1996, il existait déjà quelques solutions au problème de l'échange d'informations entre applications : sur Macintosh par exemple avec les AppleEvents ou sur Sun Solaris/UNIX avec Tooltalk[1]. Ces solutions, bien qu'adaptées à une approche événementielle conforme à nos besoins IHM, restaient cantonnées à un seul type de plateforme matérielle et logicielle. Du côté des bus logiciels on trouvait CORBA[2] et Koalatalk[3]. CORBA est trop lourd à mettre en œuvre pour des prototypes d'application écrits en langages de scripts. Par contre, KoalaTalk, développé à l'INRIA, répond assez bien aux besoins évoqués plus haut, mais son développement a été arrêté. Nous avons du choisir une autre solution et nous avons finalement décidé d'écrire notre propre bus logiciel : Ivy. Son protocole est figé depuis 1996 mais son utilisation ne cesse d'évoluer. Ivy est implémenté comme une collection de bibliothèques sous licence LGPL et fourni avec quelques agents et exemples d'utilisation selon un modèle open source sur le site http://www.tls.cena.fr/products/ivy/.

LES CHOIX DE CONCEPTION FONDAMENTAUX

Ivy possède une architecture totalement distribuée, sans serveur central, ce qui favorise le fonctionnement par agents, la flexibilité. D'autre part, comme les IHM développées sont construites essentiellement sur un modèle événementiel, Ivy se devait d'être le relais de ce type de fonctionnement en autorisant l'échange de messages simples. Enfin, par sa simplicité de mise en œuvre et d'utilisation, Ivy est pleinement compatible avec des langages de scripts comme Perl, Python ou Tcl utilisés pour la réalisation de nos prototypes d'interfaces graphiques (en utilisant la toolkit graphique Tk étendue par le widget TkZinc [4]). Des développeurs novices ne connaissant pas Ivy peuvent ainsi se connecter au bus et échanger leurs premiers messages sur le bus en moins d'une journée. Ils se familiarisent très vite avec l'outil et peuvent ainsi dialoguer avec les autres développeurs pour se consacrer aux problématiques IHM.

Le protocole ainsi que le comportement des objets utilisant Ivy ont fait l'objet d'une description [2], basée sur le formalisme des objets interactifs coopératifs [9]. Cette description permet aux concepteurs des bibliothèques de s'accorder, aux utilisateurs de comprendre quel va être le comportement de leurs programmes, permettra à terme d'avoir des moyens de tester le bon comportement des agents Ivy et de faire des tests de non régression sur les nouvelles implémentations.

[1] sunsite.ccu.edu.tw/books/books/Julienne/Julienne.html
[2] http://www.corba.org
[3] http://www-sop.inria.fr/koala/beust/koalatalk.html

PRINCIPE DE FONCTIONNEMENT D'IVY

Le principe de fonctionnement d'Ivy est des plus simples : C'est un système qui permet à des applications d'échanger des informations en ayant l'illusion de les diffuser, la sélection des informations étant laissée aux applications réceptrices. Du point de vue du programmeur, Ivy est donc un canal de diffusion où des agents envoient des messages et d'autres agents s'abonnent à la réception de certains messages. Du point de vue de l'architecture, le bus logiciel est un lieu de rendez-vous sur le réseau. C'est une adresse UDP en broadcast ou TCP multicast. Lorsqu'un programme utilisant une bibliothèque Ivy rejoint le bus, il émet sur le point de rendez-vous un message indiquant qu'il est prêt, et fournit une adresse TCP où le joindre. Les autres agents vont alors le contacter, puis échanger avec lui les coordonnées réseau et des listes d'abonnements. Pendant l'exécution des différents agents, les comportements associés aux abonnements seront exécutés chaque fois qu'un message satisfaisant aux expressions régulières est émis.

UN EXEMPLE D'UTILISATION

Nous présentons là un exemple simple d'utilisation d'Ivy en Perl.

```
# initialisation et création d'un objet ivy
Ivy->init(-loopmode => 'local');
$ivy=Ivy->new(-appName  => 'toyappli',
              -ivyBus => '235.0.0.0:4321');
…
# un abonnement
$ivy->bindRegexp("^hello (.*) time=(.*)",
                 [\&callback]);
…
…Ivy->mainLoop;
…
…
sub callback {
    my ($sendername, $part1, $part2) = @_;
    print "From $sendername : $part, $part2";
    $ivy->sendMsgs("this is my message");
}
```

Cette application se connecte au bus identifié par l'adresse de multicast '235.0.0.0:4321'. Elle s'abonne aux messages de la forme « hello xxx time=yyy » et imprime à chaque réception de ces messages la ligne suivante : « From Id zzz : xxx, yyy », où Id est le nom déclaré de l'agent qui a émis le message. Elle renvoie aussi à chaque fois le message « this is my message ».
Ivy propose également un mécanisme de notification à l'application de la connexion ou déconnexion de tout nouvel agent, par exemple pour réaliser un agent de supervision. Il offre aussi un message pour tuer et déconnecter un agent et un mécanisme d'envoi de message direct, avec un autre agent, donc sans « diffusion ».

STYLES DE PROGRAMMATION

Linton et Price ont constaté dans [5], que la programmation du comportement des objets interactifs

est souvent liée au couple formé par le langage de programmation et la boîte à outils graphique. Ils ont recensé plusieurs méthodes pour décrire cette communication et l'exécution du comportement associé, parmi lesquelles :

- L'appel direct à des méthodes d'objets connus, ce qui nécessite de savoir distinguer les différentes instances d'une même classe. Cela n'est pas adapté à la conception d'applications avec création et disparition de nombreux objets, comme une image radar,
- L'utilisation de valeurs actives, avec par exemple l'asservissement d'une jauge à une valeur numérique. Cela garantit les modifications de manière transactionnelle, mais limite la puissance d'expression en nommant et figeant la nature des valeurs qui communiqueront leurs mises à jour à leurs vues multiples,
- La propagation au sein d'un arbre ou d'un graphe, comme dans Inventor. Cela pose des problèmes de puissance d'expression lors de la conception, au bénéfice de la performance de l'affichage,
- Les mécanismes de notification implémentant en partie les schémas sujet/observateur [3]. La plupart des implémentations simplifient ces schémas au sens où il faut que l'observateur aille s'enregistrer directement auprès du sujet. Dans des implémentations comme X11 ou Java AWT, il faut s'abonner auprès d'une fenêtre, ou d'un bouton et associer un callback sous la forme d'une fonction typée

Le dernier style de programmation sied le mieux au développement de prototypes. Il est implémenté dans les langages de script et certains langages objets. Lors de la conception d'Ivy, nous avons tenté de garder le meilleur des deux mondes, c'est-à-dire de fournir aux programmeurs des abstractions du même niveau que celles qui sont présentes pour la programmation d'IHM dans le langage de son choix, avec la possibilité d'utiliser des services réseau. Une fois identifié le point de rendez-vous, on ne se soucie plus de socket ou de mainloop, on ne manipule plus que des objets directement liés au domaine que l'on programme dans un vocabulaire textuel.

RETOUR D'EXPERIENCE

Ivy joue un rôle très important dans la plupart de nos développements IHM. D'une part, Ivy joue le rôle de facilitateur car il permet de fragmenter certaines problématiques IHM, tout en favorisant la réutilisabilité. Par exemple, nous avons pu comparer facilement deux types de reconnaissance vocale (Dragon Naturally Speaking, compatible MS-API[4] sous Windows et IBM ViaVoice sous Linux avec Java Speech[5]) pour la reconnaissance automatique d'indicatifs d'avions sur la fréquence. Des tests simples ont pu se faire très rapidement en échangeant les agents de reconnaissance, sans perturber les autres agents connectés au bus et abonnés aux messages délivrés par les systèmes de reconnaissance.

Ivy permet donc de fragmenter et d'isoler les problématiques IHM en s'affranchissant des obstacles de l'ordre de la communication entre applications. Cette expérience a de nouveau été réussie lorsque nous avons voulu intégrer des appareils mobiles à notre banc de démonstration. Ces derniers, reliés au réseau local par le réseau sans fil, disposent des bibliothèques Ivy adéquates. Ils sont connectés aux bus de la même façon que les agents résidants sur le banc, ce qui permet par exemple d'interagir à distance ou de répartir l'interaction.

D'autre part, Ivy permet de fournir un point de rendez-vous pour les programmeurs, de laisser définir un langage commun plus proche des idées et des concepts que d'un consensus autour des problématiques d'implémentation de chacun. Ce fonctionnement permet aux agents de se connecter dynamiquement ou de se déconnecter sans affecter les autres. Dès lors, il est possible d'imaginer l'échange d'informations relatives à l'interaction entre les différents agents qui peuvent aussi bien être des moteurs de fusion multimodaux. Cette utilisation du bus logiciel se détache singulièrement du bus logiciel comme simple canal de transmission de données.

BIBLIOTHEQUES DISPONIBLES ET PRINCIPAUX AGENTS

Notre banc de démonstration rassemble une partie de nos prototypes pour la future position de contrôle aérien. Il est entièrement basé sur Ivy. Les bibliothèques sont en effet disponibles sur de nombreuses plateformes matérielles. Cela nous permet d'utiliser conjointement des développements sur différentes architectures suivant nos besoins. Par exemple, une image radar développée en OpenGL sous Windows est pilotée par un outil de rejeu de trafic développé en Ada sur PC Linux et manipulée grâce à une interface graphique de saisie sur écran tactile, en Perl/Tk ou depuis un PDA sous Linux, relié au réseau sans fil local.

D'autre part, les bibliothèques Ivy étant disponibles pour de nombreux langages tels que C et C++, Perl, Perl/Tk et Tcl, Java, CAML, Python, Ada, en liant la bibliothèque C et enfin en VBA (Visual Basic for Application) pour connecter les applications de MS-Office.

[4] http://www.microsoft.com/speech/

[5] http://java.sun.com/products/java-media/speech/

Il nous ainsi est possible d'adapter nos besoins aux potentialités offertes par les APIs. Ce type d'utilisation mène à la programmation de nombreux agents qui forment maintenant une collection de composants interactifs réutilisables. Ainsi, il existe aujourd'hui une trentaine d'agents capables d'échanger de l'information sur le bus et qui peuvent se ranger dans les quatre catégories suivantes :

Des agents génériques
Ce sont des agents d'aide à la mise au point ou de surveillance des messages circulant sur un bus ainsi que des agents de lancement ou de supervision d'applications connectées au bus.

Des agents orientés Contrôle du Trafic Aérien
Du fait du domaine d'application du CENA, de nombreux agents sont orientés contrôle du trafic aérien. On peut citer plusieurs images radar, des simulateurs de trafic (qui simulent le comportement de nombreux avions), des IHM comme DigiStrips [7], ou DigiBook [8], qui expérimente le Pick and Drop entre PDA, et position de travail avec écran tactile.

Des agents « grand public »
Depuis quelques mois, nous avons commencé à connecter des applications telles que Flight Simulator 2002 ou les applications du pack Office de Microsoft. Cela nous permet de fusionner les vols simulés par nos simulateur de trafic avec un ou plusieurs vols pilotés par un utilisateur de FS2002. Dans le second cas, cela permet par exemple de piloter une présentation PowerPoint à la voix, par le geste sur un tableau interactif ou depuis une ardoise électronique.

Des agents multimodaux
Comme nous avons pu le voir plus haut, l'architecture d'Ivy nous sert également de base pour le développement de systèmes multimodaux utilisant des moteurs de fusions d'événements de haut niveau transmis sur le bus par les applications graphiques, vocales et autres périphériques comme les télécommandes. Par exemple, nous avons pu tester les performances d'un système de synthèse vocale en l'utilisant depuis des conversations de l'IRC (Internet Relay Chat).

CONCLUSION ET DISCUSSION

Depuis cinq ans, l'utilisation d'Ivy au sein de notre équipe a profondément marqué les orientations de nos développements en privilégiant la réutilisabilité et la simplicité de mise en œuvre. Ivy, malgré ses limites, est un compromis très adapté aux problématiques IHM et de programmation. La multiplication des agents développés par d'autres équipes devrait permettre d'enrichir encore le champ des possibilités offertes par ce bus logiciel pour l'IHM.

Cette évolution, en favorisant la complémentarité des agents, devrait également permettre d'explorer quelques pistes de réflexions concernant l'architecture de systèmes interactifs répartis et multimodaux. Pour ces derniers en effet, Ivy pourrait dater les messages émis afin qu'ils puissent être traités par un moteur de fusion.

BIBLIOGRAPHIE

1. B.W. Boehm A spiral model of software development and enhancement, IEEE Computer 21, 5, 1988.
2. Stéphane Chatty, Yannick Jestin, Sébastien Maury, The Ivy software protocol. http://www.tls.cena.fr/products/ivy/documentation/
3. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Elements of reusable object-oriented software, Addison Wesley, 1994.
4. Patrick Lecoanet. Zinc, an advanced scriptable Canvas. available as free software on http://www.openatc.org/
5. Mark Linton, Chuck Price, Building distributed user interfaces with Fresco. The X Resource, 1993.
6. D.J. Mayhew, chapitre 1, The usability engineering lifecycle, Morgan Kaufmann, San Fransisco, 1999.
7. Christophe Mertz, Stéphane Chatty, Jean-Luc Vinot. Pushing the limits of ATC user interface design beyond S&M interaction: the DigiStrips experience. *3rd USA/Europe ATM E&D Seminar*, Napoly, 2000.
8. Marcellin Buisson, Yannick Jestin, Design issues in distributed interaction supporting tools : mobile devices in an ATC working position, MobileHCI 2001.
9. David Navarre, Philippe Palanque, Rémi Bastide, Ousmane Sy, Structuring Interactive Systems Specifications for Executability and Prototypability. 7th Eurgraphics Workshop on Design, Specification and Verification of Interactive Systems, Springer, 2000.